



# MPI + OpenMP Correctness Checking with MUST

Debugging, Testing and Correctness Workshop Series 2023

Joachim Jenke ([jenke@itc.rwth-aachen.de](mailto:jenke@itc.rwth-aachen.de))

## How many issues can you spot in this tiny example?

```
#include <mpi.h>
#include <stdio.h>
int main (int argc, char** argv)
{
    int rank, size, buf[8];

    MPI_Comm_rank (MPI_COMM_WORLD, &rank);
    MPI_Comm_size (MPI_COMM_WORLD, &size);

    MPI_Datatype type;
    MPI_Type_contiguous (2, MPI_INTEGER, &type);

    MPI_Recv (buf, 2, MPI_INT, size - rank, 123, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    MPI_Send (buf, 2, type, size - rank, 123, MPI_COMM_WORLD);
    printf ("Hello, I am rank %d of %d.\n", rank, size);

    return 0;
}
```

*At least 8 issues in this code example!*

- MPI programming is error prone
- Portability errors (just on some systems, just for some runs)
- Bugs may manifest as:
  - Crash
  - Application hanging
  - Finishes
- Questions:
  - Why crashing/hanging?
  - Is my result correct?
  - Will my code also give correct results on another system?
- Tools help to pin-point these bugs



# Must detects deadlocks

Who?

What?

Where?

Details

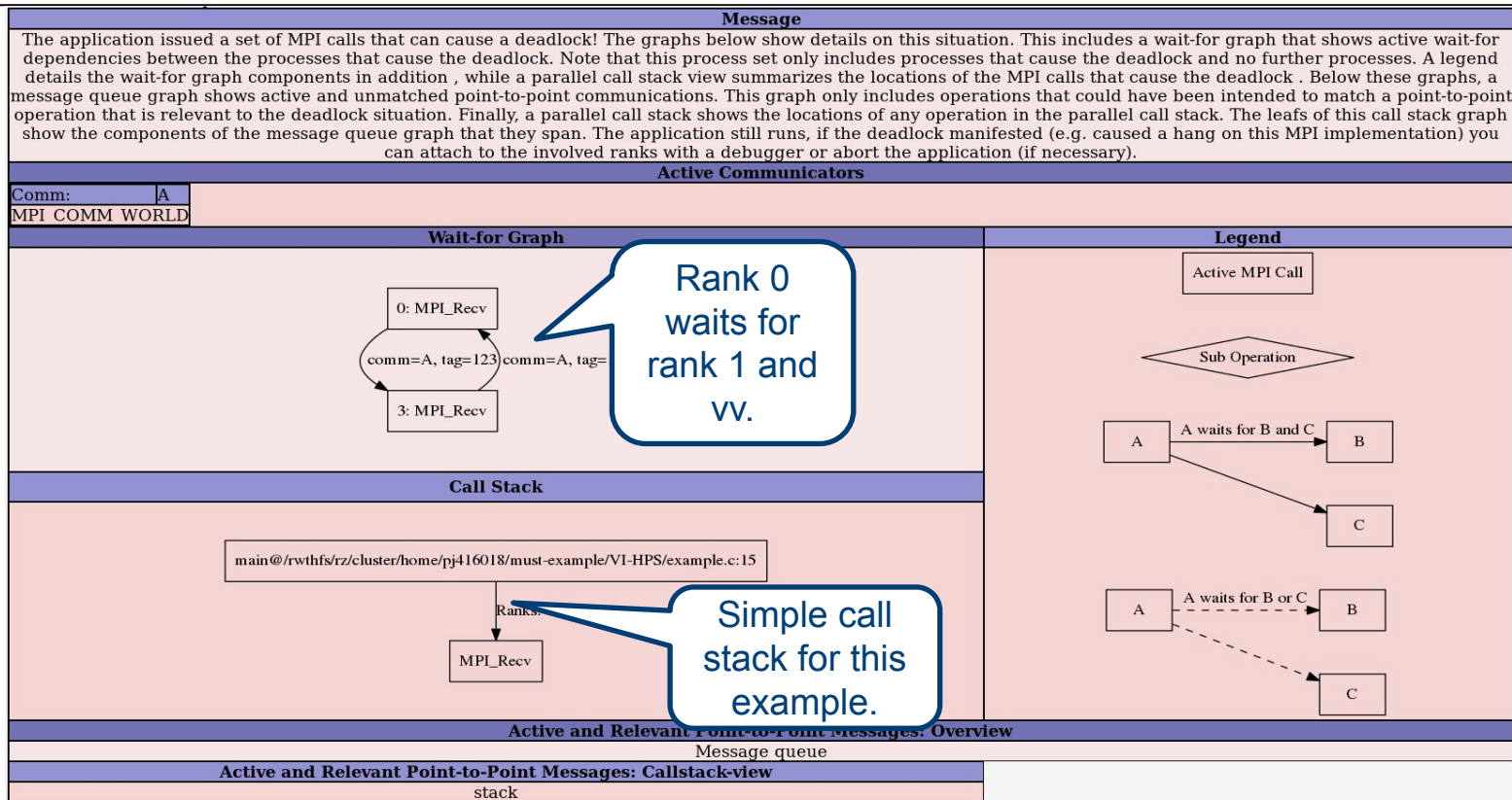
MUST Output starting date: Fri Mar 24 11:59:41 20...

Rank(s)	Type	Message
	Error	The application issued a set of MPI calls that can cause a deadlock! A graphical representation of this situation is available in <a href="#">detailed de...</a>
Details:		
Message		From References
The application issued a set of MPI calls that can cause a deadlock! A graphical representation of this situation is available in a <a href="#">detailed deadlock view (MUST Output-files/MUST_Deadlock.html)</a> . References 1-2 list the involved calls (limited to the first 5 calls, further calls may be involved). The application still runs, if the deadlock manifested (e.g. caused a hang on this MPI implementation) you can attach to the involved rank with a debugger or abort the application (if necessary).		References of a representative process:
		reference 1 rank 0: <b>MPI_Recv</b> (1st occurrence) called from: #0 main@example.c:15  reference 2 rank 3: <b>MPI_Recv</b> (1st occurrence) called from: #0 main@example.c:15

Click for graphical representation of the detected deadlock situation.



# Visualization of deadlock situation



## MUST detects errors in transfer buffer sizes / types

Rank(s)	Type	Message	From	References
2(28793)	<b>Error</b>	A receive operation uses a (datatype, count) pair that can not hold the data transferred by the send it matches! The first element of the send that did not fit into the receive operation is at (contiguous)[0](MPI_INTEGER) in the send type (consult the MUST manual for a detailed description of datatype positions). The send operation was started at reference 1, the receive operation was started at reference 2. (Information on communicator: MPI_COMM_WORLD) (Information on send of count 2 with type:Datatype created at reference 3 is for Fortran, based on the following type(s): { MPI_INTEGER}) (Information on receive of count 2 with type:MPI_INT)	Representative location: <b>MPI_Send</b> (1st occurrence) called from: #0 main@example-fix1.c:18	References of a representative process:  reference 1 rank 2: <b>MPI_Send</b> (1st occurrence) called from: #0 main@example-fix1.c:18  reference 2 rank 1: <b>MPI_Irecv</b> (1st occurrence) called from: #0 main@example-fix1.c:16  reference 3 rank 2: <b>MPI_Type_contiguous</b> (1st occurrence) called from: #0 main@example-fix1.c:13
1(28792)	<b>Error</b>	A receive operation uses a (datatype,count) pair that can not hold the data transferred by the send it matches! The first element of the send...		
0-3	<b>Error</b>	Argument 3 (datatype) is not committed for transfer, call MPI Type commit before using the type for transfer!(Information on datatypeData...		
2(28793)	<b>Error</b>	The memory regions to be transferred by this send operation overlap with regions spanned by a pending non-blocking receive operation!(In...		
1(28792)	<b>Error</b>	The memory regions to be transferred by this send operation overlap with regions spanned by a pending non-blocking receive operation!(In...		
3(28795)	<b>Error</b>	The memory regions to be transferred by this send operation overlap with regions spanned by a pending non-blocking receive operation!(In...		
3(28795)	<b>Error</b>	A receive operation uses a (datatype,count) pair that can not hold the data transferred by the send it matches! The first element of the send...		
0(28794)	<b>Error</b>	The memory regions to be transferred by this send operation overlap with regions spanned by a pending non-blocking receive operation!(In...		
0(28794)	<b>Error</b>	A receive operation uses a (datatype,count) pair that can not hold the data transferred by the send it matches! The first element of the send...		

Size of sent message  
larger than receive  
buffer

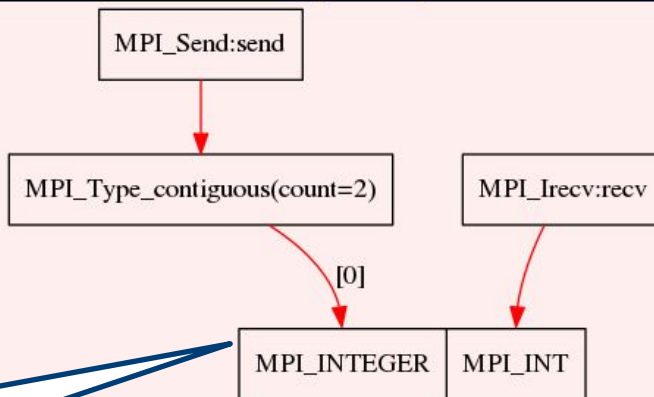
All detected errors are  
collapsed for overview  
- click to expand

## MUST detects errors in handling datatypes

### Message

The application issued a set of MPI calls that mismatch in type signatures! The graph below shows details on this situation. The first differing communication request is highlighted.

### Datatype Graph



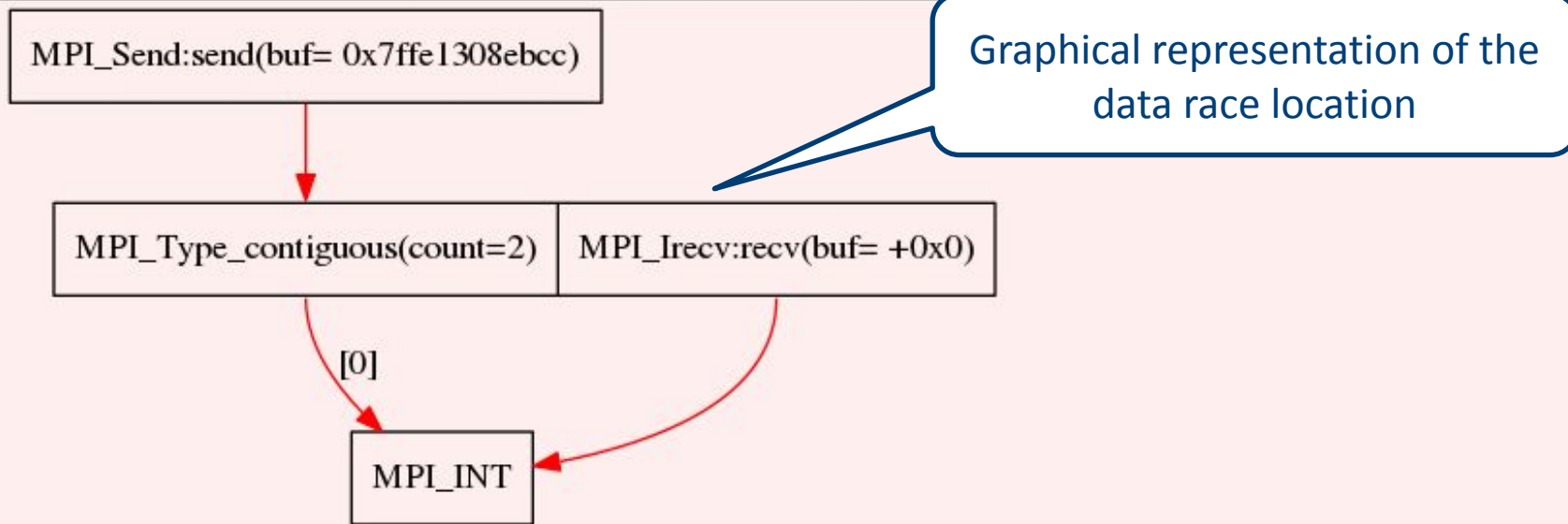
Graphical representation of the type mismatch

## Graphical representation of the race condition

### Message

Overlap in communication buffers! The graph below shows details on this situation. The first colliding communication request is highlighted.

### Datatype Graph





## MUST detects leaks of user defined objects

Rank(s)	Type	Message
0-3	Error	There are 1 datatypes that are not freed when MPI Finalize was issued, a quality application should free all MPI resources before calling ...
Details:		
Message	From	References
<p>There are 1 datatypes that are not freed when MPI Finalize was issued, a quality application should free all MPI resources before calling MPI_Finalize. Listing information for these datatypes:</p> <p>-Datatype 1: Datatype created at reference 1 is for C, committed at reference 2, based on the following type(s): { MPI_INT }</p>	<p>Representative location:  <b>MPI_Type_contiguous</b> (1st occurrence) called from:  #0 main@example-fix4.c:13</p>	<p>References of a representative process:</p> <p>reference 1 rank 1: <b>MPI_Type_contiguous</b> (1st occurrence) called from:  #0 main@example-fix4.c:13</p> <p>reference 2 rank 1: <b>MPI_Type_commit</b> (1st occurrence) called from:  #0 main@example-fix4.c:14</p>
0-3	Error	There are 1 requests that are not freed when MPI Finalize was issued, a quality application should free all MPI resources before calling M...
Details:		
Message	From	References
<p>There are 1 requests that are not freed when MPI_Finalize was issued, a quality application should free all MPI resources before calling MPI_Finalize. Listing information for these requests:</p> <p>-Request 1: Point-to-point request activated at reference 1</p>	<p>Representative location:  <b>MPI_Irecv</b> (1st occurrence) called from:  #0 main@example-fix4.c:17</p>	<p>References of a representative process:</p> <p>reference 1 rank 1: <b>MPI_Irecv</b> (1st occurrence) called from:  #0 main@example-fix4.c:17</p>

- User defined objects include
  - MPI\_Comms (even by MPI\_Comm\_dup)
  - MPI\_Datatypes
  - MPI\_Groups

Unfinished non-blocking receive is resource leak and missing synchronization

Leak of user defined datatype object

# Finally

Rank(s)	Type	Message
	Information	MUST detected no MPI usage errors nor any suspicious behavior during this application run.
Details:		
Message		FromReferences
MUST detected no MPI usage errors nor any suspicious behavior during this application run.		

No further error  
detected

Hopefully this message  
applies to many  
applications

## MUST - Basic Usage

- Apply MUST as an mpiexec wrapper, that's it:

```
% mpicc source.c -o exe  
% $MPIRUN -n 4 ./exe
```

```
% mpicc -g source.c -o exe  
% mustrun --must:mpiexec $MPIRUN -n 4 ./exe
```

or simply

```
% mustrun -n 4 ./exe
```

- After run: inspect “MUST\_Output.html”
- “mustrun” (default config.) uses an extra process:
  - I.e.: “mustrun -np 4 ...” will use 5 processes
  - Allocate the extra resource in batch jobs!
  - Default configuration tolerates application crash; BUT is slower (details later)

---

# Hands-on 1

## Hands-on 1

```
$ module use ~dc-prot1/.modules  
$ module load clang_comp/13.0.0 intel_mpi/2020-update2  
must/1.9.2/clang_13.0.0_intel_mpi_2020-update2
```

- Clone <https://git-ce.rwth-aachen.de/hpc-public/must-tutorial.git>
- Run MUST on the the example code in the Hands-on-1 directory

```
module use ~dc-prot1/.modules
```

```
module load clang_comp/13.0.0 intel_mpi/2020-update2  
must/1.9.2/clang_13.0.0_intel_mpi_2020-update2
```

```
mpigcc -g example.c
```

```
mustrun -np 2 ./a.out
```

To receive the MUST html report by mail add  
`--must:output-email-report <addr>`

To get the report on command line add  
`--must:output stdout`



---

# Advanced Usage



	Application might crash	Application never crashes
Centralized analysis		<code>--must:nocrash</code>
	<ul style="list-style-type: none"><li>▪ 1 extra process</li><li>▪ Blocking communication</li></ul>	<ul style="list-style-type: none"><li>▪ 1 extra process</li><li>▪ Non-blocking communication</li></ul>
Distributed analysis	<code>--must:nodesize 8</code>	<code>--must:fanin 8</code>
	<ul style="list-style-type: none"><li>▪ 1 extra process per 7 application processes + tree</li><li>▪ Nodesize must be divisor of ranks sharing memory</li></ul>	<ul style="list-style-type: none"><li>▪ 1 extra process per 8 app processes + tree</li></ul>

## MUST - Multithreading Support

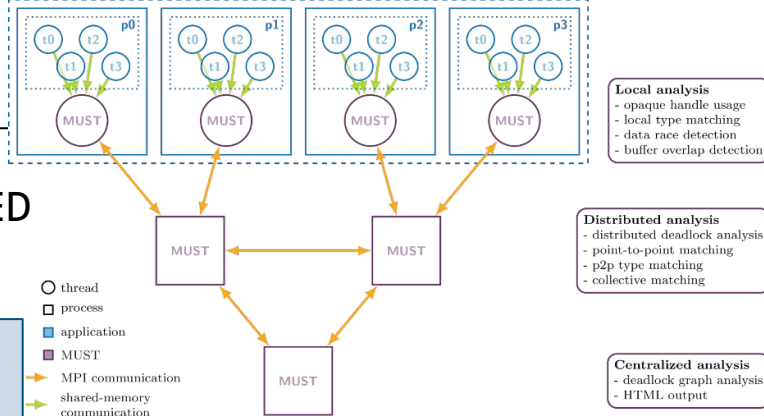
- By default, MUST supports `MPI_THREAD_FUNNELED`
- For higher threading levels:

```
% mustrun -n 40 ./exe --must:hybrid
```

- This will raise the required level to `MPI_THREAD_MULTIPLE`!
- Some MPI might need env like: `MPICH_MAX_THREAD_SAFETY=multiple`
- Get info about the resources needed:

```
% mustrun -n 40 ./exe --must:hybrid --must:info
```

➤ This will give you the number of processes needed with tool attached



## MPI runtime correctness checking with MUST – Advanced Usage

- We use Backward-cpp (or Dyninst) as an external lib for stacktraces
- Collecting stack traces can be costly. Select with  
`--must:stacktrace [backward|dyninst|none]`
- Supposed your application has no faults you won't need stacktraces 😊

Rank(s)	Type	Message	From	References
	Information	MUST detected no MPI usage errors nor any suspicious behavior during this application run.		

### From

Representative location:  
**MPI\_Init\_thread** (1st occurrence) called from:  
#0 MAIN\_@bt.f:90  
#1 main@bt.f:319

Representative location:  
**MPI\_Comm\_split** (1st occurrence) called from:  
#0 MAIN\_@bt.f:90  
#1 main@bt.f:319



## MUST – Filter file

---

- Use filter files to selectively exclude error/warning messages (avoid cluttered output)
- Format: `messageType:MUST_MESSAGE_TYPE:source`
  - `MUST_MESSAGE_TYPE`: kind of message to ignore (e.g. `MUST_WARNING_COMM_NULL`)
  - `source`: specific file (`filename.c`), specific function (`function_name`) or all sources (`*`)
- Example: Ignore NULL comm. warnings originating from `main.c` (needs stacktraces)

```
messageType:MUST_WARNING_COMM_NULL:src:main.c
```

- Example: Ignore all data type leak errors

```
messageType:MUST_ERROR_LEAK_DATATYPE:*
```

- Define and use a filter file:
  - `--must:filter-file <path-to-filter-file>`

## MUST – More options

---

- Print help:
  - `--must:help`
- Select output format:
  - `--must:output {html|json|stdout}`
- Use with ddt:
  - Record error message information:
    - `--must:capture`
  - Replay under control of ddt:
    - `--must:reproduce --must:ddt`

## Tool Overview - Approaches Techniques

- Debuggers:
  - ✓ Helpful to pinpoint any error
  - Finding the root cause may be hard
  - Won't detect sleeping errors
  - E.g.: gdb, TotalView, Allinea DDT
- Static Analysis:
  - Compilers and Source analyzers
  - ✓ Typically: type and expression errors
  - E.g.: MPI-Check
- Model checking:
  - ✓ Can find hidden errors
  - Requires a model of your applications
  - State explosion possible
  - E.g.: MPI-Spin

```
MPI_Recv (buf, 5, MPI_INT,  
         -1,  
         123, MPI_COMM_WORLD, &status);
```

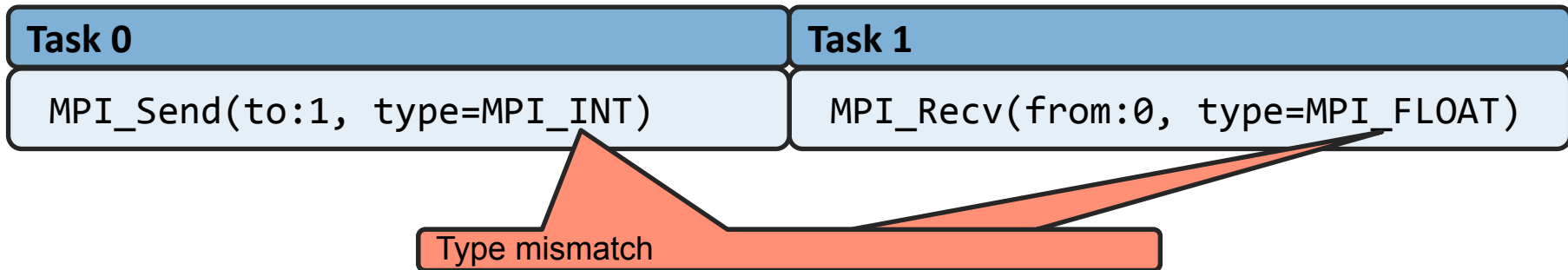
“-1” instead of “MPI\_ANY\_SOURCE”

```
if (rank == 1023)  
    abort ();
```

Only works with less than 1024 tasks

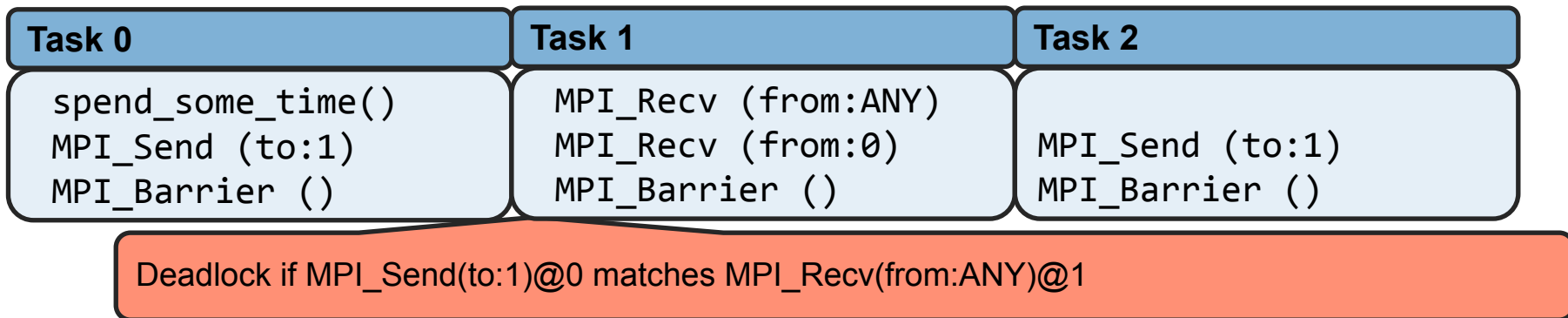
## Tool Overview - Approaches Techniques (2)

- Runtime error detection:
  - ✓ Inspect MPI calls at runtime
    - Limited to the timely interleaving that is observed
    - Causes overhead during application run
    - E.g.: Intel Trace Analyzer, Umpire, Marmot, MUST



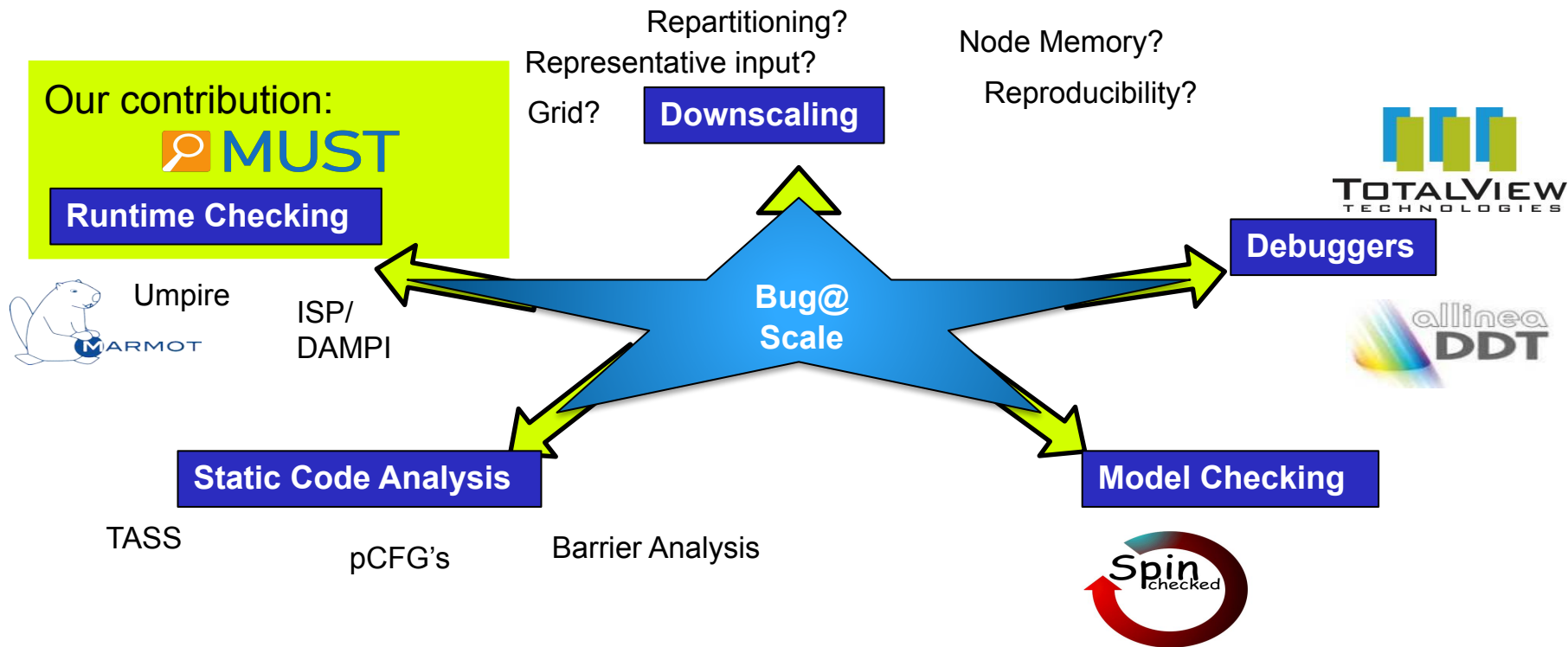
## Tool Overview - Approaches Techniques (3)

- Formal verification:
  - Extension of runtime error detection
  - Explores all relevant interleavings (explore around nondet.)
  - Detects errors that only manifest in some runs
  - Possibly many interleavings to explore
  - E.g.: ISP





## Approaches to Remove Bugs (Selection)



## MUST - Summary

---

- MPI runtime error detection tool
- Open source (BSD license)  
<http://www.itc.rwth-aachen.de/MUST/>
- Wide range of checks, strength areas:
  - Overlaps in communication buffers
  - Errors with derived datatypes
  - Deadlocks
- Largely distributed, able to scale with the application

---

# Compiler-Aided Analysis

## How many issues can you spot in this tiny example?

```
#include <mpi.h>
#include <stdio.h>
int main (int argc, char** argv)
{
    int rank, size, buf[8], provided = 0, requested = MPI_THREAD_MULTIPLE;
    MPI_Init_thread (&argc, &argv, MPI_THREAD_SINGLE, &provided);
    printf ("provided: %d.\n", provided);
    MPI_Comm_rank (MPI_COMM_WORLD, &rank); MPI_Comm_size (MPI_COMM_WORLD, &size);
    MPI_Request request;
    #pragma omp parallel sections num_threads(2)
    {
        MPI_Send (buf, 4, MPI_FLOAT, size - rank - 1, 123, MPI_COMM_WORLD); // thread 0
    #pragma omp section
        MPI_Irecv (buf, 4, MPI_FLOAT, size - rank - 1, 123, MPI_COMM_WORLD, &request); // thread 1
    }
    MPI_Wait (&request, MPI_STATUS_IGNORE);
    MPI_Finalize();
    return 0;
}
```

At least 3 issues in this code example!

- Compile the MPI(+OpenMP) application just like described for Archer
- Using clang to compile with OpenMPI/IntelMPI/MPICH:
  - export `OMPI_CC=clang`; export `MPICH_CC=clang`;
  - export `OMPI_CXX=clang++`; export `MPICH_CXX=clang++`;
- Run MUST with TSan support:
  - `--must:tsan`
- For integration of TSan output into the MUST report, a helper-library must be linked into the application (not necessary for the setup on cosma):
  - `-Wl,--whole-archive ${MUST_ROOT}/lib/libonReportLoader.a`
  - `-Wl,--no-whole-archive`

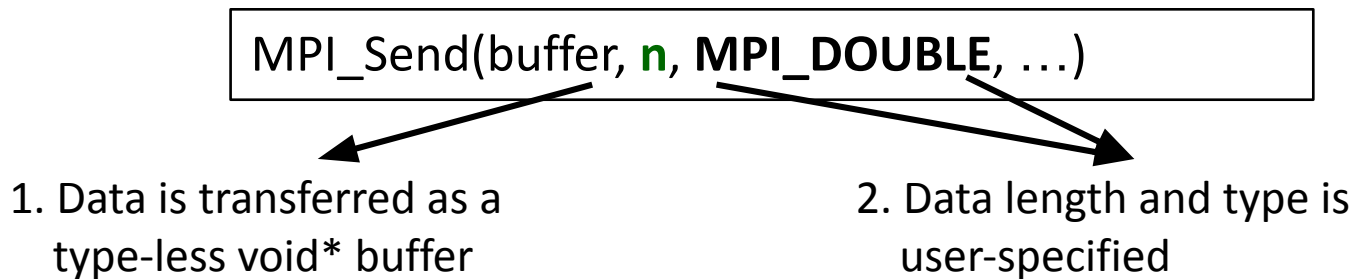


## TSan Output in MUST report

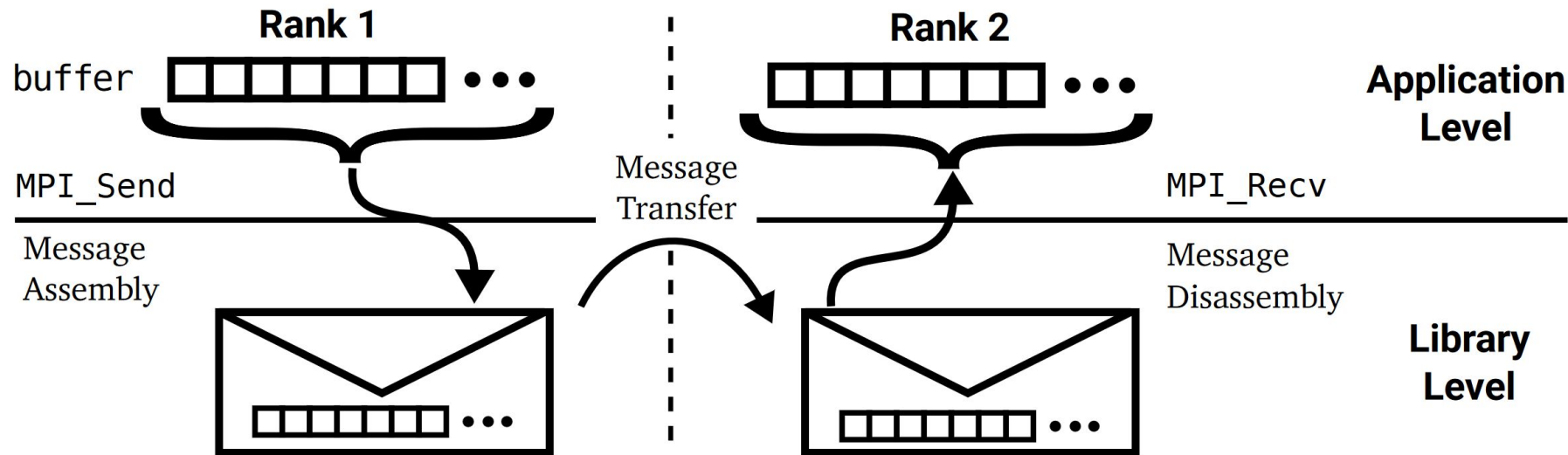
Rank(s)	Type	Message
0-7	MUST_WARNING_DATA_RACE	Data race between a read of size 8 at .omp_outlined._debug_.53@1 and a previous write of size 8 at .omp_outlined._debug...
Details:		
Message	From	References
Data race between a read of size 8 at .omp_outlined._debug_.53@1 and a previous write of size 8 at .omp_outlined._debug_.53@2.	<p>Representative location:</p> <p><b>.omp_outlined._debug_.53</b> (0th occurrence) called from:</p> <p>#0 .omp_outlined._debug_.53@lulesh.cc:2258</p> <p>#1 .omp_outlined..54@lulesh.cc:2240</p> <p>#2 __kmp_invoke_microtask@libomp.so:0xbad72</p> <p>#3 EvalEOSForElems(Domain&amp;, double*, int, int*, int)@lulesh.cc:2240</p> <p>#4</p> <p>ApplyMaterialPropertiesForElems(Domain&amp;)@lulesh.cc:2401</p> <p>#5 LagrangeElements(Domain&amp;, int)@lulesh.cc:2439</p> <p>#6 LagrangeLeapFrog(Domain&amp;)@lulesh.cc:2617</p> <p>#7 main@lulesh.cc:2748</p>	<p>References of a representative process:</p> <p>reference 1 rank 1: <b>.omp_outlined._debug_.53</b> (0th occurrence) called from:</p> <p>#0 .omp_outlined._debug_.53@lulesh.cc:2258</p> <p>#1 .omp_outlined..54@lulesh.cc:2240</p> <p>#2 __kmp_invoke_microtask@libomp.so:0xbad72</p> <p>#3 EvalEOSForElems(Domain&amp;, double*, int, int*, int)@lulesh.cc:2240</p> <p>#4 ApplyMaterialPropertiesForElems(Domain&amp;)@lulesh.cc:2401</p> <p>#5 LagrangeElements(Domain&amp;, int)@lulesh.cc:2439</p> <p>#6 LagrangeLeapFrog(Domain&amp;)@lulesh.cc:2617</p> <p>#7 main@lulesh.cc:2748</p> <p>reference 2 rank 1: <b>.omp_outlined._debug_.53</b> (0th occurrence) called from:</p> <p>#0 .omp_outlined._debug_.53@lulesh.cc:2246</p> <p>#1 .omp_outlined..54@lulesh.cc:2240</p> <p>#2 __kmp_invoke_microtask@libomp.so:0xbad72</p> <p>#3 EvalEOSForElems(Domain&amp;, double*, int, int*, int)@lulesh.cc:2240</p> <p>#4 ApplyMaterialPropertiesForElems(Domain&amp;)@lulesh.cc:2401</p> <p>#5 LagrangeElements(Domain&amp;, int)@lulesh.cc:2439</p> <p>#6 LagrangeLeapFrog(Domain&amp;)@lulesh.cc:2617</p> <p>#7 main@lulesh.cc:2748</p> <p>#8 main@lulesh.cc:2715</p>

MPI libraries provide only minimal error checking

- Buffers are passed as void\*
- At the same time, users have to specify buffer length and the datatype manually
  - Error-prone!

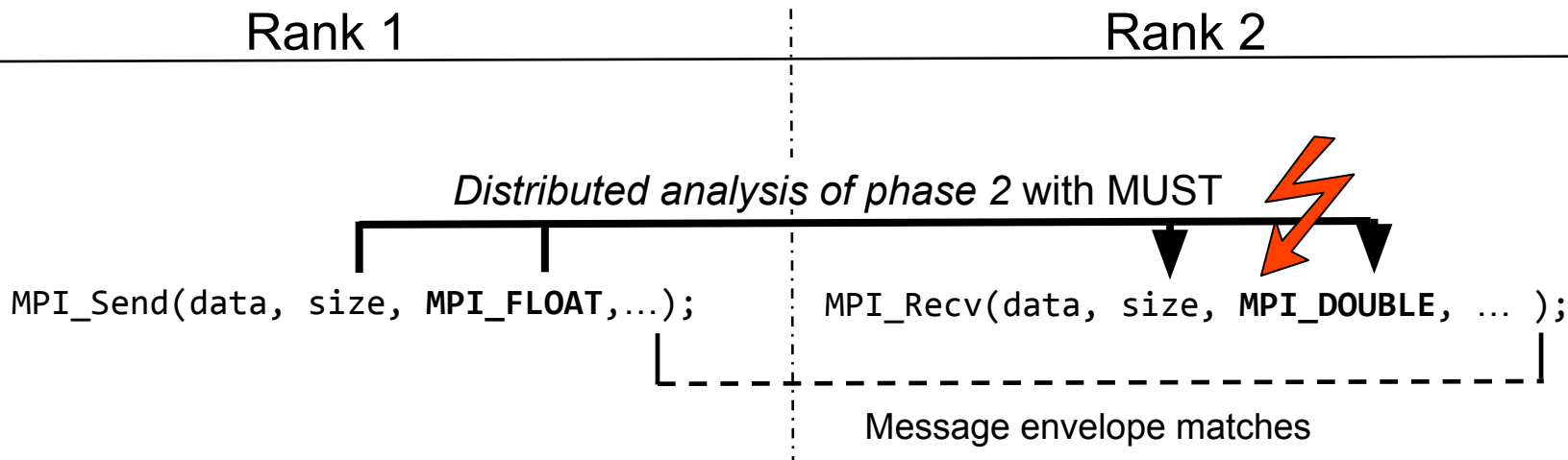


1. Data is pulled out of the (send) buffer for message assembly
2. Data is transferred from sender to receiver
3. Data is disassembled and put into the (receive) buffer



Adapted from <https://dl.acm.org/doi/10.1145/3431379.3460652>

A message transfer from sender (rank 1) to receiver (rank 2)



MUST needs to check type-less void\*  
buffer **data**

- Phase 1, e.g., MPI\_Send
- Phase 3, e.g., MPI\_Recv

Answer the questions:

- *Is it of type **MPI\_FLOAT**?*
- *Is it of length **buffer\_size**?*

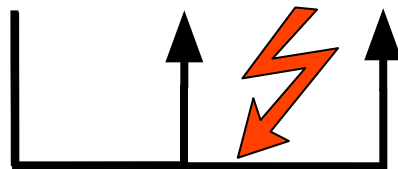
→ Can not be answered by MUST  
without further tooling

Rank 1

Memory Allocation  
**double\*** buff = ...;



MPI\_Send(data, buffer\_size, **MPI\_FLOAT**,...);



*Local analysis only  
possible with tooling*

### AMG2013, a CORAL performance and parallel scaling benchmark [Coral'20]

- In parcsr\_mv/par\_csr\_matrix.c:1236, reported by [DKL LLVM'15]:

```
MPI_Bcast( &global_data[3], global_size-3, MPI_INT, ...);
```

HYPRE\_BigInt alias long long

The diagram illustrates the mapping of the MPI\_INT type to HYPRE\_BigInt and then to long long via an alias. A black arrow points from MPI\_INT to HYPRE\_BigInt, and another black arrow labeled 'alias' points from HYPRE\_BigInt to long long. A red arrow points from the MPI\_INT in the code to the long long in the diagram.

### 104.milc, a SPEC MPI benchmark [SpecMPI'07]

- In com\_mpi.c:480:

```
MPI_Allreduce( cpt, &work, 2, MPI_FLOAT, ... );
```

struct{float a; float b;}; → Benign today, but tomorrow?

The diagram shows the value 2 in the MPI\_Allreduce function being interpreted as an array of 2 floats, represented by the struct {float a; float b;}. A red arrow points from the 2 in the code to the struct. A black arrow points from the struct to the text 'Benign today, but tomorrow?'. The struct is enclosed in a dashed box.

TypeART is a tool to track allocations and their type information

- Consists of
  - a LLVM compiler plugin to **instrument allocations** (LLVM IR level),
    - Heap, stack, global: Memory address, element count and type
  - a runtime with a C API to **provide type information** to MUST



1. Compile and link your application with TypeART compiler wrapper
  - mpicc → typeart-mpicc
  - mpicxx → typeart-mpic++

} Filters allocation if never passed to MPI
2. Replace “mpiexec” with command “mustrun” and activate TypeART, e.g.,
  - mustrun -np 4 --must:typeart ./my-app.bin
3. Inspect “MUST\_Output.html” in run directory for (type-related) issues

## make

- Often compiler selection possible with env variable

Makefile content:

```
MPICC ?= mpicc
demo: example.c
    $(MPICC) -O1 -g -c $< -o $@.o
    $(MPICC) $@.o -o $@
```

```
$> MPICC=typeart-mpicc make -j 1
```

## CMake

- During the configuration, CMake executes internal compiler checks, where we do not need TypeART instrumentation:

- ☐ Need to disable wrapper



```
$> TYPEART_WRAPPER=OFF cmake .. \
-DMAKE_C_COMPILER=typeart-mpicc
```

```
$> make -j 1
```

TypeART type-related errors may look as follows (mustrun -np 4 --must:typeart ./app)

- Here, MPI\_Irecv expects an **MPI\_INT** buffer, but a **float\*** buffer handle was passed

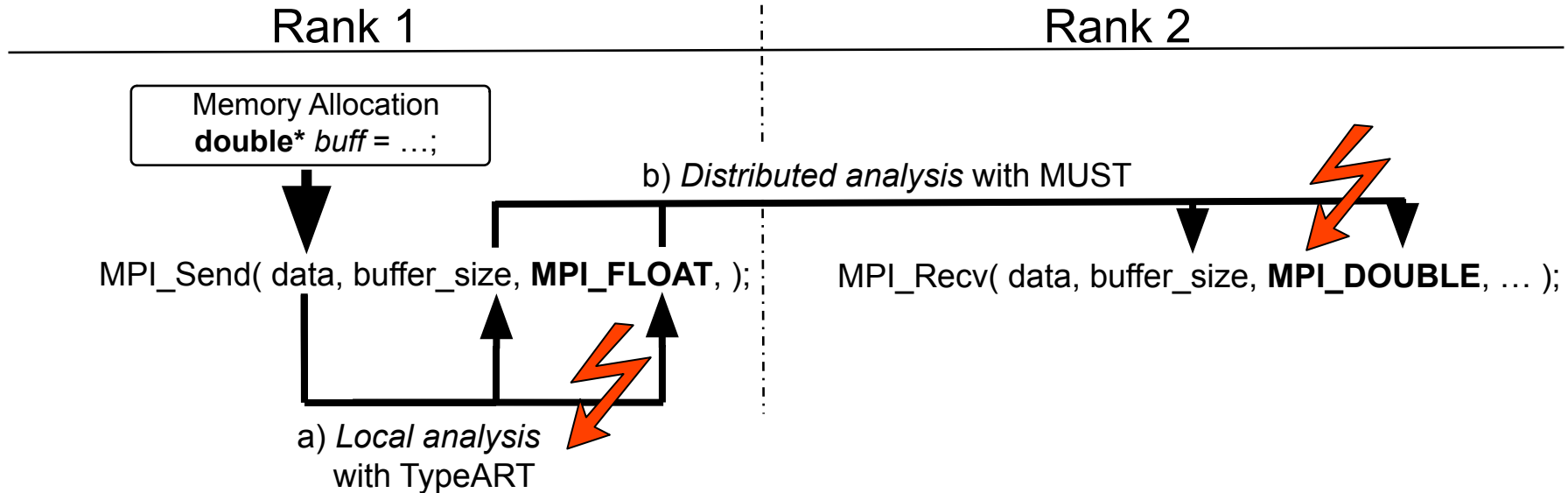
MUST Output, starting date: Sun Jun 11 17:06:48 2023.

Rank(s)	Type	Message
0-7	MUST_ERROR_TYPMATCH_MISMATCH	Incompatible buffer of type 5 (float) - expected MPI_INT instead
Details:		
Message		From
Incompatible buffer of type 5 (float) - expected MPI_INT instead		Representative location: MPI_Irecv (1st occurrence) called from: #0 CommRecv(Domain&, int, int, int, int, bool, bool)@/pc2/users/a/ahueck/nhr/lulesh/lulesh-comm.cc:119 #1 main@/pc2/users/a/ahueck/nhr/lulesh/lulesh.cc:2723
0-7	MUST_ERROR_TYPMATCH_MISMATCH	Incompatible buffer of type 5 (float) - expected MPI_INT instead
7	MUST_ERROR_TYPMATCH_MISMATCH	Incompatible buffer of type 5 (float) - expected MPI_INT instead

MUST has completed successfully, end date: Sun Jun 11 17:06:48 2023.

MUST Version: v1.9.0

MUST & TypeART combine dynamic analysis with compile-time instrumentation to enable type checking for all phases of message transfer



---

# Hands-on 2

## Hands-on 2

```
$ module use ~dc-prot1/.modules  
$ module load clang_comp/17.0.6 intel_mpi/2020-update2  
must/1.10.0-preview/clang_17.0.6_intel_mpi_2020-update2
```

- Clone <https://git-ce.rwth-aachen.de/hpc-public/must-tutorial.git>
- Run MUST on the LULESH code in the Hands-on-2 directory
- The README.md file contains instructions for this Hands-on